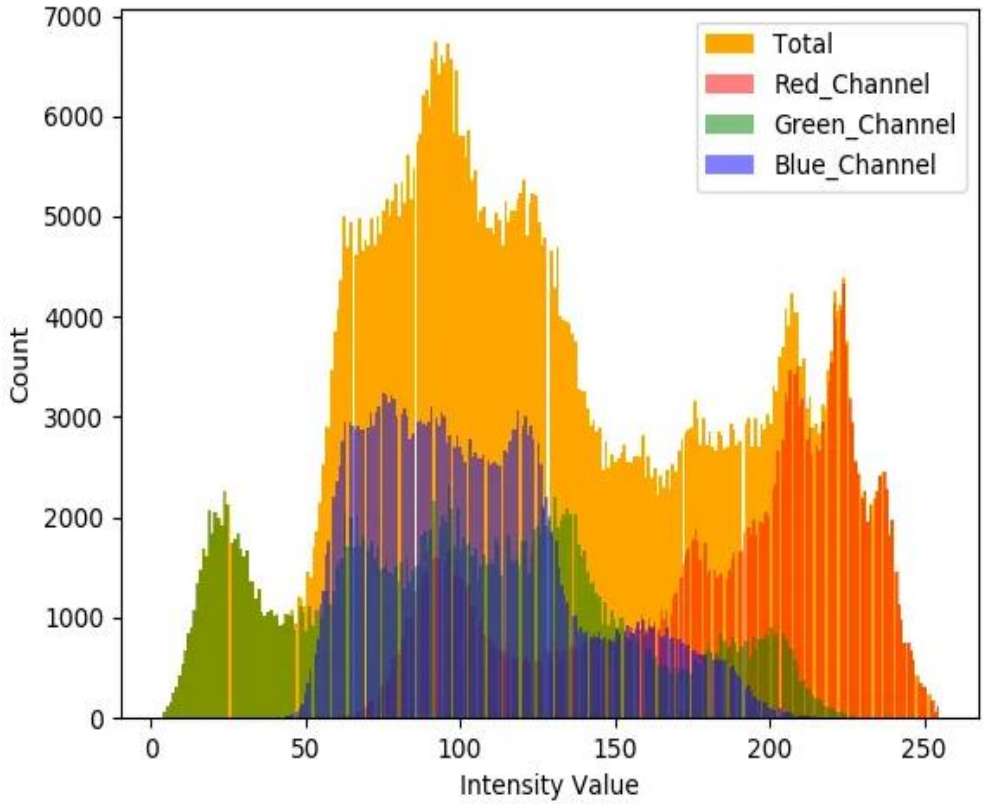
Computer Vision and Pattern Recognition CSI-7-CVP Tutorial session(week-2)

Pawel Markiewicz (pawel.markiewicz@lsbu.ac.uk)

Enrico Grisan, Faria Hossain

Histograms in Image Processing with skimage-Python

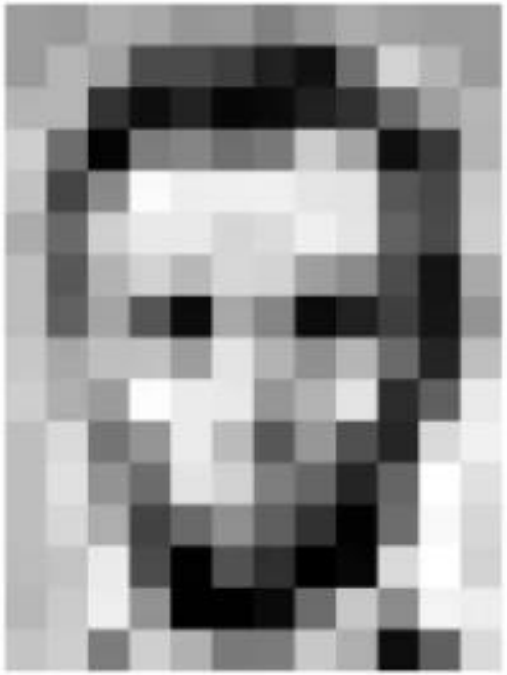


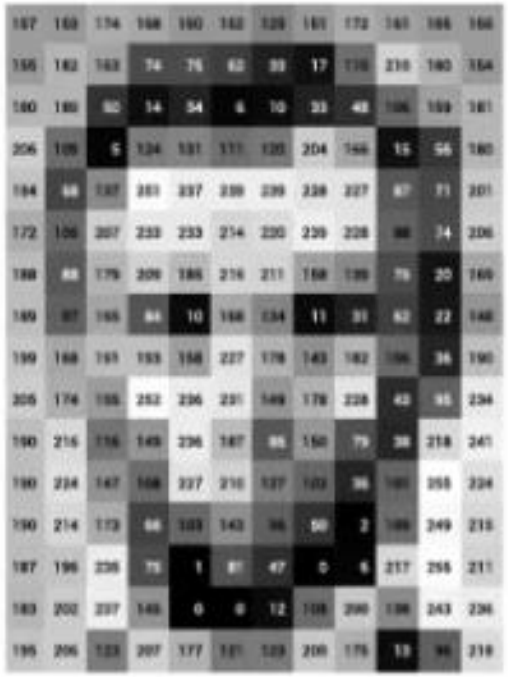
- Histograms are used to depict many aspects of the image we are working with. Such as,

- Exposure
- Contrast
- Dynamic Range
- Saturation

By visualizing the histogram we can improve the visual presence of an image and also we can find out what type of image processing could have been applied by comparing the histograms of an image.

What is a Histogram?





187	183	174	168	150	152	129	151	172	161	155	186
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Images are stored as pixel values, each pixel value represents a color intensity value. Histograms are the frequency distribution of these intensity values that occur in an image.

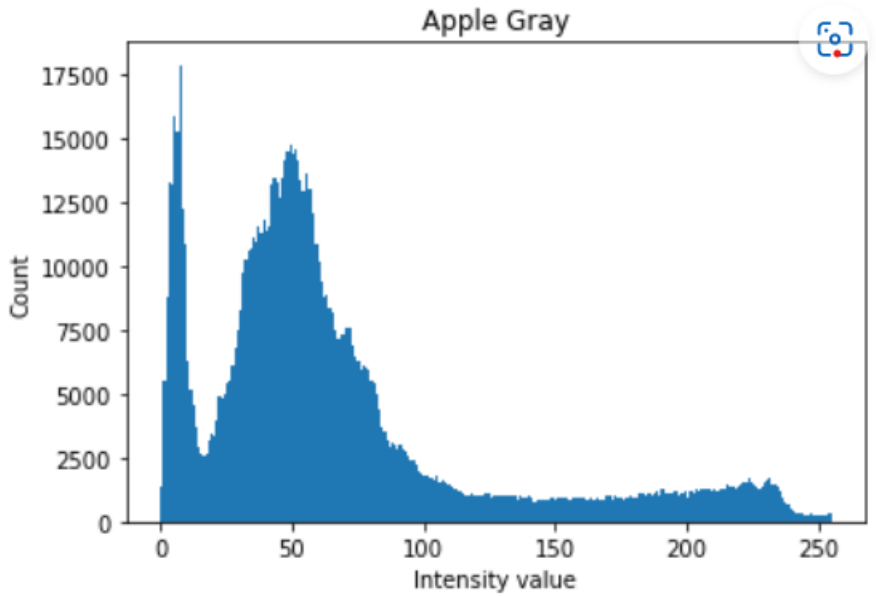
h(i) = the number of pixels in I(image) with the intensity value I

For example, if i = 0, the h(0) is the number of pixels with a value of 0.

Creating a Histogram of an Image with skimage

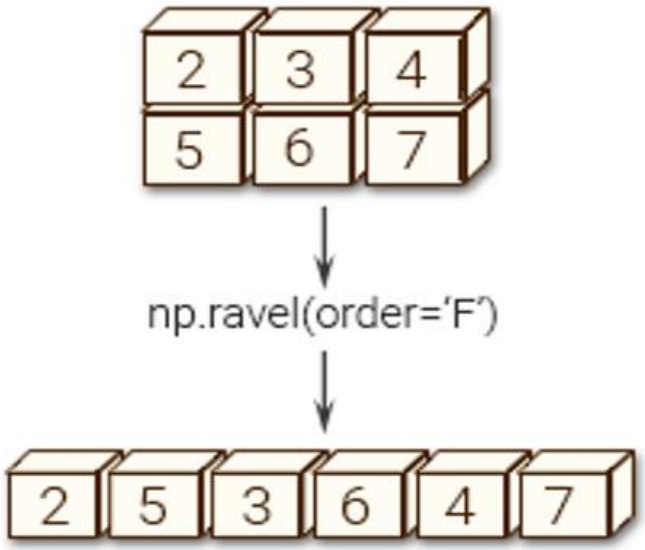
```
from skimage import io
import matplotlib.pyplot as plt

image = io.imread('apple gray.jpeg')
ax = plt.hist(image.ravel(), bins = 256)


plt.title("Apple Gray")
plt.xlabel("Intensity value")
plt.ylabel("Count")


plt.show()
```

Grayscale Image



- In the above code, we have loaded the grayscale image of an apple and generated its histogram using matplotlib. Since the image is stored in the form of a 2D ordered matrix we converted it to a 1D array using the ravel() method.

NumPy: ravel() function



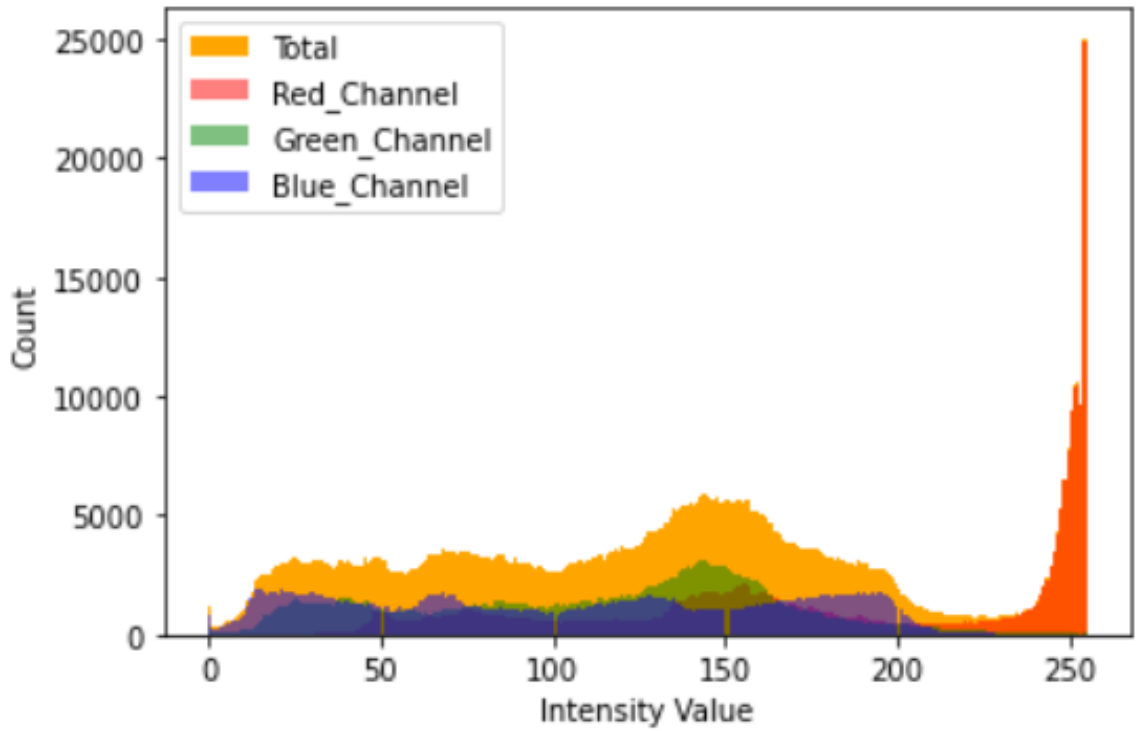
- The ravel() function is used to create a contiguous flattened array.

Colour Image

- In color images, we have 3 color channels representing RGB. In the Combined Colour Histogram, the intensity count is the sum of all three color channels.

$$h(i) = h\_red(i) + h\_green(i) + h\_blue(i)$$

```python
from skimage import io
import matplotlib.pyplot as plt
image = io.imread('flower.jpeg')

_ = plt.hist(image.ravel(), bins = 256, color = 'orange', )
_ = plt.hist(image[:, :, 0].ravel(), bins = 256, color = 'red', alpha = 0.5)
_ = plt.hist(image[:, :, 1].ravel(), bins = 256, color = 'Green', alpha = 0.5)
_ = plt.hist(image[:, :, 2].ravel(), bins = 256, color = 'Blue', alpha = 0.5)
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')
_ = plt.legend(['Total', 'Red_Channel', 'Green_Channel', 'Blue_Channel'])
plt.show()

#Output: Figure-2
```


Total
Red_Channel
Green_Channel
Blue_Channel
25000
20000
15000
10000
5000
0
Count
0
50
100
150
200
250
Intensity Value

What is Histogram Binning?

- Usually, the range of intensity values of images is from [0–255] in 8bits representation(2^8).
- But images can be also represented using 2^{16}, 2^{32} bits and so on.
- In such cases the intensity range is high and it is hard to represent each intensity value in a histogram.

- We use binning to overcome the above problem. Here we quantize the range into several buckets. For example,

if we quantize 0-255 into 8 bins, here our bins will be
0-31, 32-63, 64-95, 96-127, 128-159, 160-191, 192-223, 224-255

- Now we need to find a way to put each intensity value into the appropriate bins. We can simply solve this,

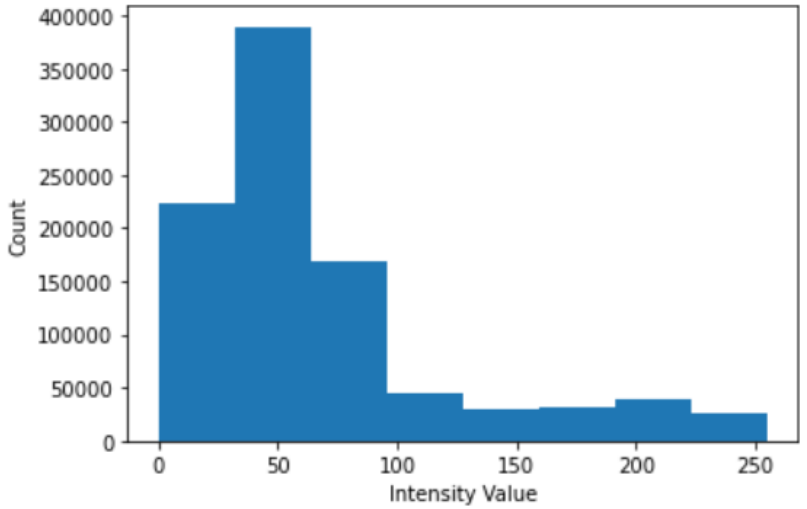
k = 256 #number of possible integer values in 8 bits representation
b = 8 #number of bins

j = floor((h(i)*b)/k)

#j is the bin number of the intensity value at position i

Plot the histogram of the grayscale apple image again but this time with 8 bins

```
from skimage import io
import matplotlib.pyplot as plt

image = io.imread('apple gray.jpeg')
_ = plt.hist(image.ravel(), bins = 8 )
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')

plt.show()
#Output: Figure-3
```



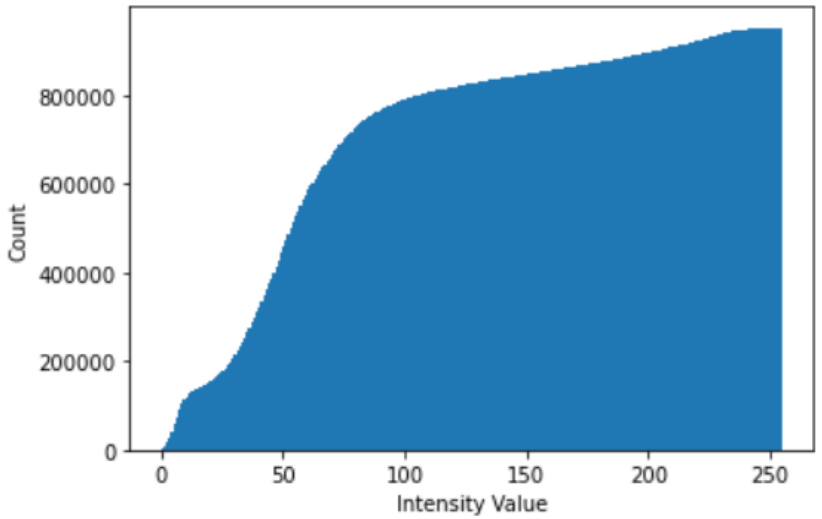
What is a Cumulative Histogram?

- The cumulative histogram is a special histogram that can be derived from the normal histogram.
- We find the counts of each intensity value from 0–255 and then add each subsequent counts,

if i = 0 then H(i) = h(0)
else H(i) = H(i-1) + h(0)

- Cumulative histograms are useful in many image processing applications like histogram equalization and so on.

```
from skimage import io
import matplotlib.pyplot as plt

image = io.imread('apple gray.jpeg')
_ = plt.hist(image.ravel(), bins = 256, cumulative = True)
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')

plt.show()
#Output: Figure-4
```



Digital Image Sampling and Quantization

- In Digital Image Processing, signals captured from the physical world need to be translated into digital form by “Digitization” Process.

- In order to become suitable for digital processing, an image function f(x,y) must be digitized both spatially and in amplitude.

Sampling: Digitizing the co-ordinate value is called sampling

Quantization: Digitizing the amplitude value is called quantization

- More specifically, we will be looking at how the spatial resolution and intensity value discretization can affect the image's overall quality.

- As usual, we import libraries such as numpy and matplotlib. Additionally, we import specific functions from the skimage library.

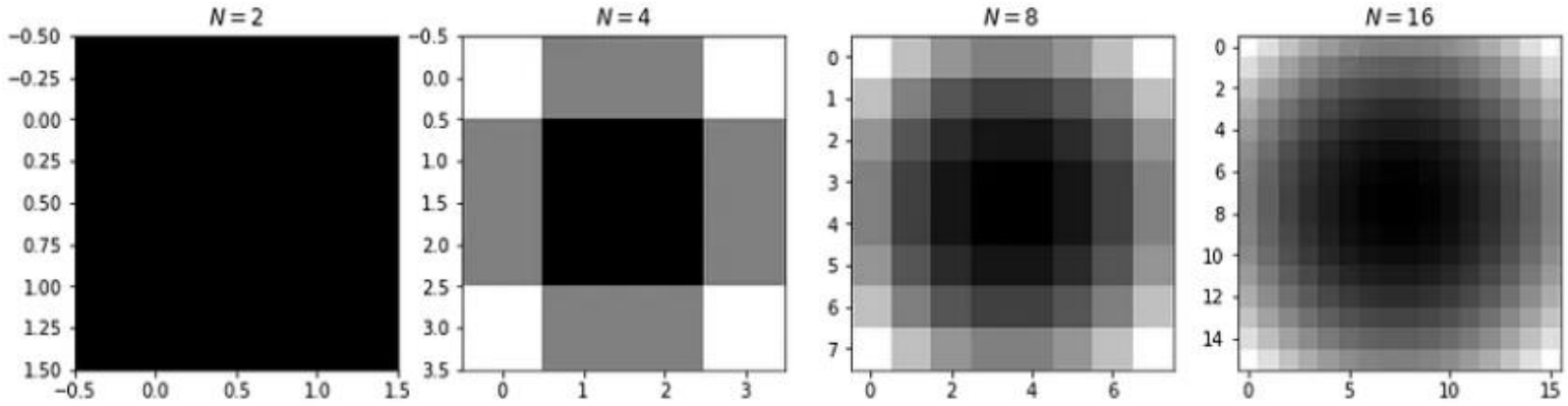
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imshow, imread
```

Image Sampling

- **Image sampling** involves taking the value of the image at regular spatial intervals. The length of the intervals defines the spatial resolution of the image.

- To better explain this, let us try to sample an analog image of a circle where N is the number of pixels on the side of the image.

```
def circle_image(x, y):
X, Y = np.meshgrid(x, y)
return X**2 + Y**2factors = 2**np.arange(1, 5)
fig, ax = plt.subplots(1, len(factors), figsize=(15, 4))
for i, N in enumerate(factors):
image = circle_image(np.linspace(-1, 1, num=N), np.linspace(-1, 1, num=N))
ax[i].imshow(image, cmap='gray')
ax[i].set_title('$N = {}$'.format(N))
```



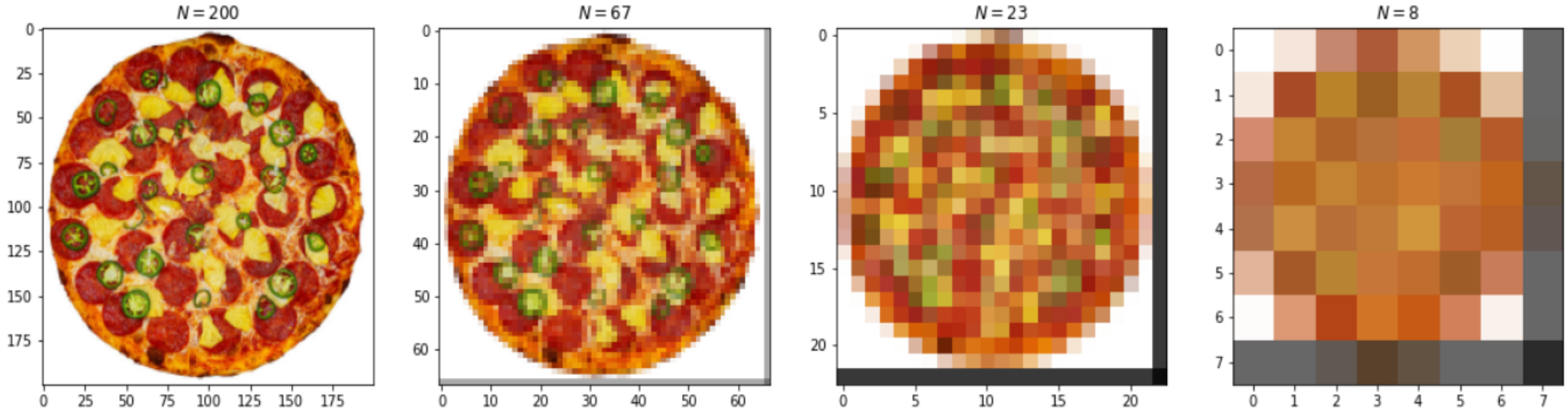
- the image’s resolution is better, and the points are much denser as N increases. This signifies that as we increase the number of pixels in the image, we can better represent an analog object into a digital image.

- The purpose of **meshgrid** is to create a rectangular grid out of an array of x values and an array of y values.

- **np.arrange** return evenly spaced values within a given interval.Values are generated within the half-open interval [start, stop) (in other words, the interval including start but excluding stop).

- Let’s try this in an actual image. Here I have an image of a pizza.

```
pizza = imread('pizza.jpg')
from skimage.transform import downscale_local_mean
factors = 3**np.arange(1, 5)
figure, axis = plt.subplots(1, len(factors), figsize=(20, 6))
for factor, ax in zip(factors, axis):
    image = downscale_local_mean(pizza,
                  factors=(factor, factor, 1)).astype(int)
    ax.imshow(image)
    ax.set_title('$N={}$'.format(image.shape[0]))
```



- In the first two images, we can observe that there is not much of a difference. This signifies that unless we need to zoom in on the image, we do not require a higher resolution on the image to understand the image's details better.

- However, as we further decrease the sampling on the image's spatial resolution, it is clear that the image is no longer clear. Lots of information is lost, and the digital image cannot fully represent the analog object.

Image Quantization

- **Image quantization** involves discretizing the intensity values of the analog image.

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

- Return evenly spaced numbers over a specified interval.
- Returns num evenly spaced samples, calculated over the interval [start, stop].
- The endpoint of the interval can optionally be excluded.

numpy.digitize(x, bins, right=False)

- Return the indices of the bins to which each value in input array belongs.

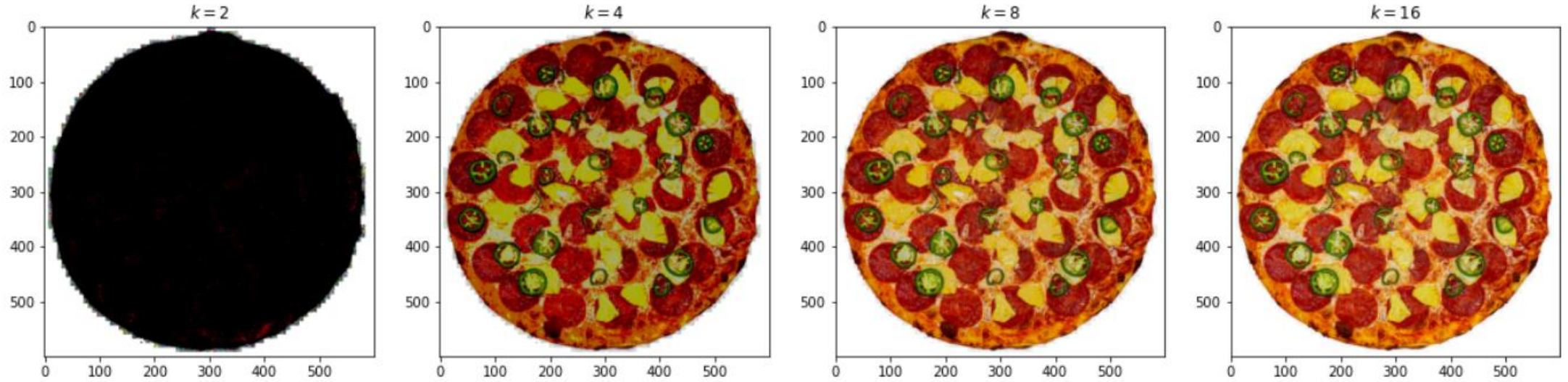
- **Numpy vectorize** function takes in a python function (pyfunc) and returns a vectorized version of the function. The vectorized version of the function takes a sequence of objects or NumPy arrays as input and evaluates the Python function over each element of the input sequence.

- The color representation of the image is much better as k increases.

- This signifies that as we increase the number of discrete values representing the image intensity values in the image, the better we can represent an analog object in a digital image.

Again, let's try this in our image of a pizza.

```
pizza = imread('pizza.jpg')
factors = 2**np.arange(1, 5)
figure, axis = plt.subplots(1, len(factors), figsize=(20, 6))
for k, ax in zip(factors, axis):
    bins = np.linspace(0, pizza.max(), k)
    image = np.digitize(pizza, bins)
    image = (np.vectorize(bins.tolist().__getitem__)
                          (image-1).astype(int))
    ax.imshow(image)
    ax.set_title('$k = {}$'.format(k))
```



- On the lower spectrum, we can see that as we further decrease the image's discretization, the image quality degradation becomes more apparent. This comes to the point where the digital image can no longer represent the analog object's varying shades of color due to the limited intensity value range.

- On the higher spectrum, we can observe that there is not much of a difference. This signifies that there is a limit on how much discrete intensity values the human eye can perceive. If we further increase the number of discrete intensity values, there will be little to no difference in the image's quality.

Summary

- We have explored how image quality can be improved and degraded using image sampling and quantization.

- Too little sampling or quantization of images can drastically degrade its quality, whereas too much can have no incremental improvement in image quality.